

Politechnika Warszawska

W Y D Z I A Ł   M A T E M A T Y K I  
I   N A U K   I N F O R M A C Y J N Y C H



# Praca dyplomowa inżynierska

na kierunku Informatyka

Interfejs użytkownika do manualnego obrysu struktur oraz wybranych  
anomalności w obrazach medycznych z wykorzystaniem tabletu  
graficznego

**Łukasz Garstecki**

Numer albumu 276857

**Tomasz Świerczewski**

Numer albumu 276915

promotor

dr inż. Magdalena Jasionowska

WARSZAWA 2019

.....

podpis promotora

.....

podpisy autorów

### **Streszczenie**

Interfejs użytkownika do manualnego obrysu struktur oraz wybranych anormalności w obrazach medycznych z wykorzystaniem tabletu graficznego

Przykładowe streszczenie. Do wykonania jako ostatnie.

**Słowa kluczowe:** slowo1, slowo2, ...



## Abstract

English title

Sample abstract in english.

**Keywords:** keyword1, keyword2, ...



Warszawa, dnia .....

### Oświadczenie

Oświadczam, że moją część pracy inżynierskiej (zgodnie z podziałem zadań opisanym na wstępie) pod tytułem „Interfejs użytkownika do manualnego obrysu struktur oraz wybranych anormalności w obrazach medycznych z wykorzystaniem tabletu graficznego”, której promotorem jest dr inż. Magdalena Jasionowska, wykonałem samodzielnie, co poświadczam własnoręcznym podpisem.

.....





# Spis treści

<b>Wstęp</b>	<b>11</b>
<b>1 Wprowadzenie</b>	<b>12</b>
1.1 Zagdania medyczne związane z aplikacją . . . . .	12
1.2 Podział prac . . . . .	12
<b>2 Stan wiedzy</b>	<b>13</b>
2.1 Przegląd istniejących rozwiązań . . . . .	13
2.1.1 Cornerstone . . . . .	13
2.1.2 DICOM Web Viewer (DWV) . . . . .	14
2.1.3 Dicom-contour . . . . .	14
2.1.4 Orthanc . . . . .	14
2.2 Proponowane rozwiązanie . . . . .	15
<b>3 Opis autorskiego systemu informatycznego</b>	<b>17</b>
3.1 Specyfikacja wymagań . . . . .	17
3.1.1 Opis biznesowy . . . . .	17
3.1.2 Wymagania funkcjonalne . . . . .	17
3.1.3 Wymagania нефункционалне . . . . .	19
3.2 Architektura rozwiązania . . . . .	19
3.3 Opracowany algorytm półautomatyczny . . . . .	19
3.3.1 Wykrycie krawędzi na bitmapie . . . . .	21
3.3.2 Stworzenie grafu z bitmapy . . . . .	23
3.3.3 Zapewnienie spójności grafu . . . . .	28
3.3.4 Wyszukanie najkrótszych ścieżek w grafie . . . . .	29
3.3.5 Optymalizacja . . . . .	32
3.4 Moduł obliczeń statystyk . . . . .	33

<b>4</b>	<b>Przeprowadzone eksperymenty</b>	<b>35</b>
4.1	Zbiór testowy . . . . .	35
4.2	Wydajność algorytmu półautomatycznego . . . . .	35
4.3	Analiza wyników i wnioski . . . . .	35
<b>5</b>	<b>Podsumowanie</b>	<b>36</b>
5.1	Napotkane problemy i ograniczenia . . . . .	36
5.2	Możliwości dalszego rozwoju . . . . .	36
	<b>Bibliografia</b>	<b>37</b>
	<b>Instrukcja instalacji</b>	<b>39</b>
	<b>Instrukcja użytkowania</b>	<b>40</b>
	<b>Wykaz symboli i skrótów</b>	<b>41</b>
	<b>Spis zawartości załączonej płyty CD</b>	<b>42</b>

## Wstęp

O czym jest praca? Co się w niej znajduje? Jaki jest wkład autora?

## 1. Wprowadzenie

### 1.1. Zagdanienia medyczne związane z aplikacją

### 1.2. Podział prac

## 2. Stan wiedzy

W poniższym rozdziale zostały przedstawione istniejące na rynku aplikacje oferujące podobne funkcjonalności do wymagań postawionych przed autorskim systemem do obrysów na obrazach DICOM. Przedstawiono również proponowane rozwiązanie postawionych przed systemem wymagań.

### 2.1. Przegląd istniejących rozwiązań

#### 2.1.1. Cornerstone

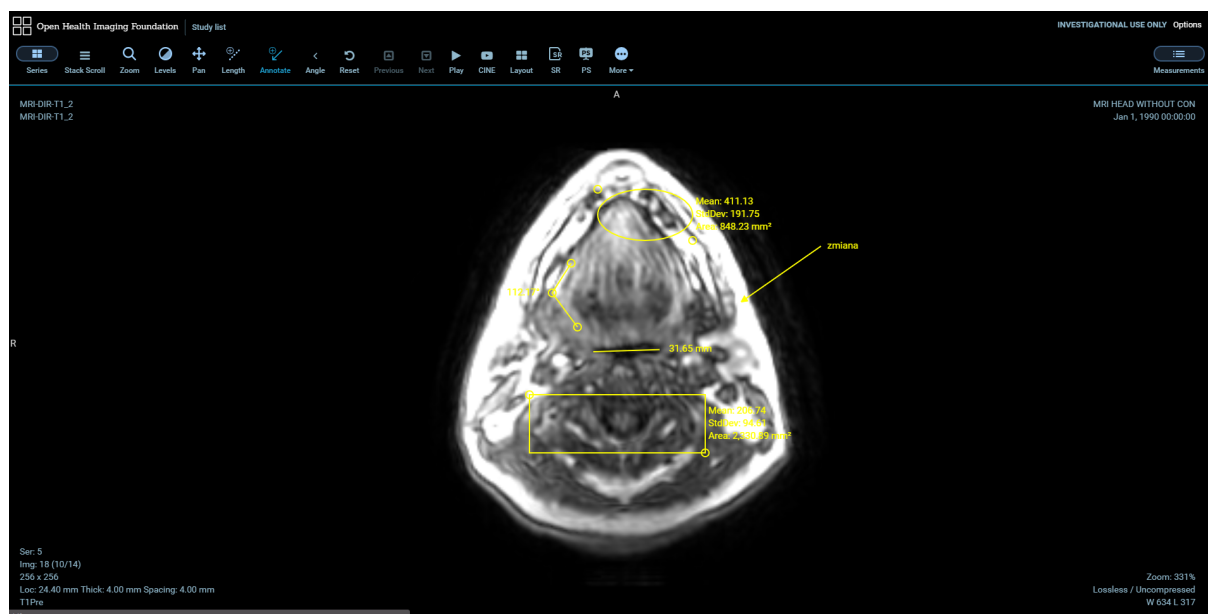
Cornerstone Core to biblioteka, która umożliwia wyświetlanie obrazów medycznych z wykorzystaniem elementu canvas z języka HTML5. Biblioteka udostępnia interfejs do wyświetlania obrazów medycznych pozwalający na zarządzanie wyświetlaniem zdjęcia. Podstawowe funkcjonalności obsługiwane przez bibliotekę to:

- Przybliżanie i oddalanie obrazu.
- Obrót obrazu.
- Przesuwanie obrazu w wyświetlanym komponencie.
- Zmiana jasności wyświetlanego obrazu.
- Mapowanie kolorów.
- Interpolacja pikseli w obrazie (dla obrazów o niskiej rozdzielczości).

Ponadto twórcy biblioteki Cornerstone Core stworzyli bibliotekę Cornerstone Tools, która korzysta z biblioteki Cornerstone Core i umożliwia wiele funkcjonalności potrzebnych lekarzom do analizy badań pacjentów. Poza funkcjonalnościami Cornerstone Core umożliwia także:

- Mierzenie odległości w linii prostej na obrazie z podaniem rzeczywistych wartości.
- Oznaczanie obszarów przy pomocy prostokątów oraz elips.
- Oznaczanie niewielkich zmian w postaci małego okręgu.
- Mierzenie kątów na podstawie 3 wybranych przez użytkownika punktów.

Przykładowym projektem korzystającym z bibliotek Cornerstone jest OHIF<sup>1</sup> Viewer. Aplikacja pozwala na wykorzystanie większości możliwości udostępnianych przez biblioteki Cornerstone. Przykładowe użycie aplikacji zostało przedstawione na rysunku 2.1.



Rysunek 2.1: Przykład użycia aplikacji OHIF Viewer

### 2.1.2. DICOM Web Viewer (DWV)

### 2.1.3. Dicom-contour

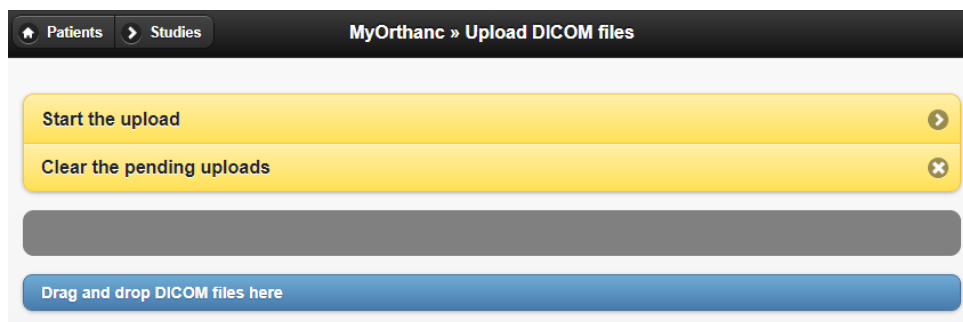
### 2.1.4. Orthanc

Orthanc to serwer z bazą plików DICOM, który umożliwia łatwe przechowywanie, zarządzanie oraz dostęp do plików medycznych DICOM. Ponadto Orthanc udostępnia REST API, które umożliwia przeglądanie wgranych na serwer plików DICOM podzielonych względem pacjentów, badań i serii.

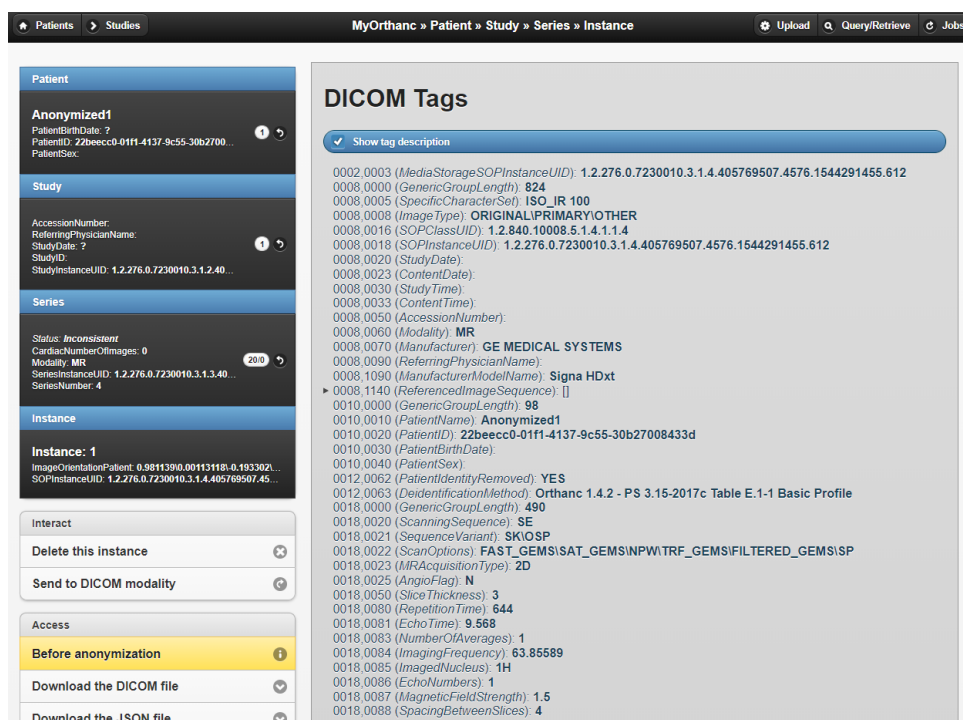
Serwer Orthanc udostępnia również prosty interfejs przeglądarkowy, który pozwala na wgrywanie nowych plików (interfejs wgrywania plików przedstawiono na rysunku 2.2), przeglądanie zapisanych plików — w szczególności tagów (rysunek 2.3) oraz podglądu obrazu (rysunek 2.4). Interfejs przeglądarkowy nie zapewnia żadnej metody rysowania na przeglądany obrazie. Pozwala natomiast na przełączanie obrazu na kolejny przez kliknięcie lewym przyciskiem myszy w prawą część podglądu obrazu.

<sup>1</sup>Open Health Imaging Foundation - <http://ohif.org/>

## 2.2. PROPONOWANE ROZWIĄZANIE

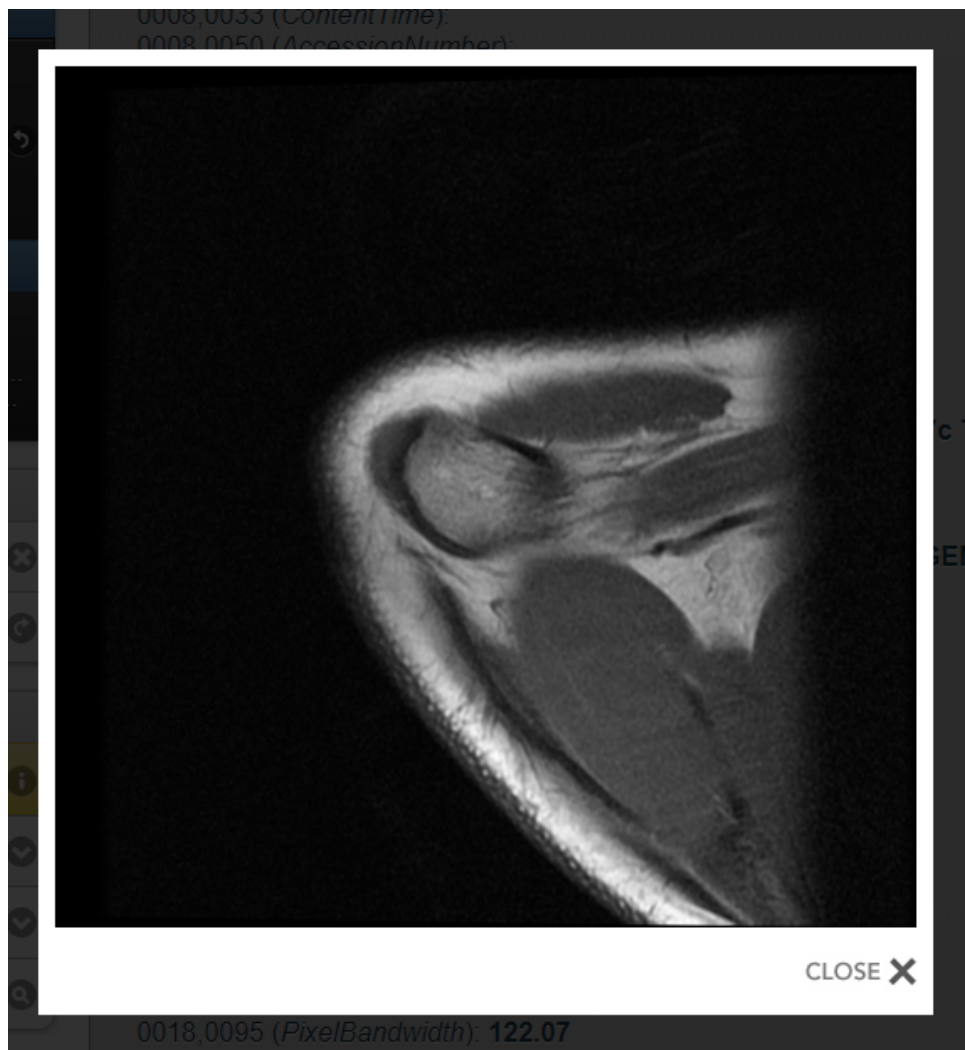


Rysunek 2.2: Interfejs wgrywania plików DICOM do serwera Orthanc



Rysunek 2.3: Podgląd tagów pliku DICOM przez interfejs przeglądarkowy Orthanc

## 2.2. Proponowane rozwiązanie



Rysunek 2.4: Podgląd obrazu DICOM przez interfejs przeglądarkowy Orthanc



### 3. Opis autorskiego systemu informatycznego

W poniższym rozdziale zawarto dokumentację techniczną i biznesową tworzonego systemu. Przedstawiono w szczególności: wymagania, architekturę, zastosowane metody półautomatycznego obrysu oraz metody obliczania statystyk obrysu.

#### 3.1. Specyfikacja wymagań

##### 3.1.1. Opis biznesowy

Celem projektu jest stworzenie interfejsu przyjaznego użytkownikowi, który umożliwi przeglądanie plików DICOM, a także przeprowadzanie na tych plikach obrysów. Prace obejmują stworzenie aplikacji webowej, która udostępni użytkownikowi interfejs komunikujący się z bazą danych Orthanc oraz serwera odpowiedzialnego za przechowywanie wygenerowanych przez użytkownika obrysów oraz wyznaczanie obrysów półautomatycznych.

Do podstawowych funkcjonalności systemu zaliczają się:

- Generowanie obrysu manualnego.
- Generowanie obrysu półautomatycznego na podstawie punktów wybranych przez użytkownika.
- Zapisywanie wygenerowanych obrysów.
- **Anonimizacja**<sup>1</sup> danych zapisanych w strukturze pliku DICOM.

##### 3.1.2. Wymagania funkcjonalne

Poniżej przedstawiono wymagania funkcjonalne w postaci historyjek użytkownika (ang. user stories):

---

<sup>1</sup> Anonimizacja (ang. anonymization) — operacja mająca na celu usunięcie z danych informacji o pacjentach, które pozwoliłyby na identyfikację danych z tożsamością pacjenta. Są to między innymi: imiona, nazwisko, pesel. Inne tłumaczenia słowa anonymization — utajnianie, usuwanie danych niejawnych. Z uwagę na fakt, że te tłumaczenia nie oddają dobrze kontekstu, zastosowano kalkę językową.

**1. Jako użytkownik chcę wczytać obraz DICOM.**

Użytkownik może wybrać obraz w menu bocznym, w którym ma możliwość wyboru pacjenta, badania oraz serii. Wybranie serii skutkuje wyświetleniem pierwszego obrazu DICOM z tej serii.

**2. Jako użytkownik chcę zmienić obraz w serii przy użyciu rolki myszy.**

Po umieszczenie kursora na obrazie przewijanie rolką myszy do góry powoduje zmianę wyświetlanego obrazu na kolejny obraz w serii. Gdy przewijamy rolką myszy do góry na ostatnim obrazie w serii wyświetlany obraz nie zmienia się. Analogicznie przewijanie rolką myszy w dół powoduje zmianę wyświetlanego obrazu na poprzedni obraz w serii, a przewijanie w dół rolką myszy na pierwszym obrazie w serii nie powoduje zmiany obrazu.

**3. Jako użytkownik chcę wykonać obrys przy użyciu tabletu graficznego.**

Po umieszczenie kursora na obrazie sterowanym przez tablet graficzny, użytkownik prowadzi kursor po obrazie wykonując obrys bez odrywania końcówki rysika od podkładki. Jeżeli użytkownik nie zakończy obrysu dokładnie w punkcie, w którym go rozpoczął, obrys powinien zakończyć się linią prostą, łączącą punkt końcowy z punktem początkowym.

**4. Jako użytkownik chcę wygenerować obrys na podstawie wybranych punktów.**

Po umieszczenie kursora na obrazie użytkownik może wybierać punkty, na podstawie których zostanie wygenerowany obrys, poprzez wciśnięcie lewego przycisku myszy w miejscach, w których chce, aby znalazły się punkty. Użytkownik może zobaczyć efekt wygenerowanego przez system obrysu.

**5. Jako użytkownik chcę edytować listę punktów, z której wygenerowany zostanie obrys.**

Użytkownik może usunąć wcześniej wybrany punkt po umieszczając nad nim kursor i wciśnięciu lewego przycisku myszy. Użytkownik może dodać nowy punkt do listy punktów poprzez wciśnięcie lewego przycisku myszy w miejscu, w którym chce wstawić punkt.

**6. Jako użytkownik chcę wybrać kolor obrysu.**

Użytkownik wybiera kolor z palety kolorów lub może zdefiniować własny kolor poprzez podanie kodu RGB koloru, który chce wybrać.

**7. Jako użytkownik chcę zapisać obrys.**

Po wykonaniu obrysu manualnego lub wybraniu listy punktów do wygenerowania obrysu półautomatycznego, użytkownik wybiera nazwę obrysu i zapisuje obrys w systemie.

#### 8. Jako użytkownik chcę obejrzyć zapisany obrys.

Użytkownik wybiera z listy po prawej stronie zapisany obrys i przegląda obrys naniesiony na obraz, na którym został wykonany.

#### 9. Jako użytkownik chcę zobaczyć statystyki dotyczące obrysu.

Użytkownik wybiera z listy po prawej stronie zapisany obrys i przegląda statystyki obliczone na podstawie zapisanego obrysu. Do statystyk zalicza się obwód obrysu, pole obrysu, histogram obrazu na obszarze obrysu oraz liczba pikseli wewnątrz obrysu.

#### 10. Jako użytkownik chcę zobaczyć jednocześnie dowolną liczbę zapisanych w systemie obrysów na jednym obrazie DICOM.

Użytkownik wybiera poprzez kliknięcie lewym przyciskiem myszy na nazwie obrysu znajdującej się na liście po prawej stronie. Wybrane obrysy wyświetlane są jednocześnie na przeglądającym przez użytkownika zdjęciu. Użytkownik może wyłączyć podgląd wcześniej wybranego obrysu poprzez ponowne wciśnięcie lewego przycisku myszy na nazwie obrysu na liście po prawej stronie. Na zdjęciu wyświetlane są jedynie obrysy wykonane na tym obrazie.

#### 11. Jako użytkownik chcę zanonimizować dane pacjenta zawarte w pliku DICOM.

Użytkownik może zanonimizować pacjenta, gdy przegląda jego obraz. Użytkownik może anonimizować imię i nazwisko pacjenta, datę urodzenia pacjenta oraz płeć pacjenta poprzez nadanie nowych wartości lub poprzez usunięcie poprzedniej wartości i pozostawienie pustych pól w formularzu.

#### 3.1.3. Wymagania нефункциональные

Tabela 3.1 przedstawia wymagania нефункциональные, które system musi spełnić.

### 3.2. Architektura rozwiązania

#### 3.3. Opracowany algorytm półautomatyczny

Opracowany algorytm półautomatyczny służy do wykrywania krawędzi na obrazie medycznym. Jest algorytmem półautomatycznym, ponieważ jest wspomagany przez człowieka — użytkownika, który wybiera punkty na ekranie. Te punkty są interpolowane przez algorytm półautomatyczny, zwany dalej algorytmem.

Jako dane wejściowe do algorytmu uzyskujemy następujące informacje:

Tablica 3.1: Spis wymagań niefunkcjonalnych

Obszar wymagań	Nr wymaga- nia	Opis
Użyteczność (ang. Usability)	1	Każda funkcjonalność aplikacji dostępna dla użytkownika musi mieścić się na pojedynczym ekranie przy rozdzielczości 1920x1080 i czcionce nie mniejszej niż 12pt.
	2	Aplikacja powinna udostępniać pobranie zapisanych obrysów przy użyciu serwisu REST.
Niezawodność (ang. Reliability)	3	Aplikacja ma być dostępna 24h w ciągu doby. Dopuszczalny jest brak działania aplikacji w dowolnym momencie przez okres nie dłuższy niż przez 12h. Po przerwie w działaniu aplikacja musi być dostępna przez kolejne 24h bez utrudnień.
Wydajność (ang. Performance)	4	Aplikacja powinna pobierać dane zewnętrzne w postaci pliku DICOM (około 20MB) nie dłużej niż 5 sekund.
	5	Aplikacja powinna generować obrys półautomatyczny i zapisywać obrys do systemu w czasie nie dłuższym niż 30 sekund.
	6	Aplikacja powinna reagować na działanie użytkownika (z wyłączeniem generowania obrysu półautomatycznego i zapisu obrysu do systemu) w czasie nie dłuższym niż 1 sekunda.

### 3.3. OPRACOWANY ALGORYTM PÓŁAUTOMATYCZNY

- Identyfikator obrazu medycznego, na którym był wykonywany obrys.
- Lista punktów wybranych przez użytkownika. Punkty te zostały wcześniej przeskalowane ze współrzędnych w aplikacji internetowej (aplikacji webowej, ang. web application) na współrzędne odpowiadające rozdzielczości obrazu medycznego.

Algorytm można podzielić na kilka ważnych etapów:

- Wykrycie krawędzi na bitmapie,
- Stworzenie grafu z bitmapy,
- Zapewnienie spójności grafu,
- Wyszukanie najkrótszych ścieżek w grafie.

Poniżej zostaną przedstawione dokładne rozwiązania dla każdego z tych kroków. Przed rozpoczęciem przetwarzania jest pobierany obraz medyczny o danym wcześniej identyfikatorze ze serwera Orthanc. Jest on podstawą do dalszej pracy algorytmu.

#### 3.3.1. Wykrycie krawędzi na bitmapie

Często na obrazach medycznych różnice w charakterystyce poziomów szarości pikseli reprezentujących interesujące nas obiekty są małe, nie są dane dodatkowe informacje o naturze obrazu. Problem opracowania uniwersalnego algorytmu wykrywania krawędzi jest problemem trudnym. Dla wielu algorytmów można znaleźć takie przykłady, że te algorytmy nie wyznaczają poprawnie krawędzi. Te algorytmy muszą spełniać szereg wymagań, które stwierdzają poprawność danego algorytmu, czy też operatora morfologicznego. Zgodnie z [4] „dobry detektor krawędzi powinien spełniać następujące warunki:

- niskie prawdopodobieństwo zaznaczenia punktów nienależących do krawędzi oraz niskie prawdopodobieństwo niezaznaczenia punktów należących do krawędzi,
- zaznaczone punkty krawędzi powinny być możliwie blisko jej osi,
- wyłącznie jedna odpowiedź na pojedynczy punkt krawędzi.”

Po zapoznaniu się z literaturą związaną z przetwarzaniem obrazów medycznych w algorytmie został użyty operator Canny’ego. Jest on powszechnym i dobrze sprawdzonym rozwiązaniem do wykrywania krawędzi. Jak napisał autor [4] „Operator ten (Canny’ego) jest bardzo popularny, chętnie wykorzystywany i adoptowany do wielu zastosowań. (...) Stał on się również standardem często używanym do porównań innych metod wykrywania krawędzi.” Zgodnie z rysunkiem 4.25 „Porównanie operatorów wykrywania krawędzi” w [4] najlepiej wykrywał główne narządy, takie jak wątroba czy też trzustka. Z wyżej wymienionych powodów został on wykorzystany w tym algorytmie.

Operator Canny’ego [5] składa się z 3 zasadniczych kroków:

#### 1. Określenie wartości i kąta gradientu.

W tym celu został wykorzystany operator gradientu, a estymatorem gradientu w funkcji dyskretniej, jaką jest obraz, zastosowano maskę, czy też operator Sobela. Wykorzystując go uzyskano dla każdego piksela wielkość oraz kierunek gradientu, co służy do dalszych obliczeń.

#### 2. Wykrycie miejsc występowania krawędzi.

W tym celu został wykorzystany algorytm non-max suppression. Polega on na wyborze takich pikseli, które mają największą wartość gradientu na linii o kierunku zgodnym z kątem danego gradientu. Możliwe są 4 kierunki: pionowy, poziomy oraz dwa diagonalne. Jeśli dany piksel miał większą wartość gradientu od dwóch swoich sąsiadów, to zaznaczono go jako potencjalny punkt tworzący krawędzie. W ten sposób otrzymano obraz z potencjalnymi krawędziami.

#### 3. Wyznaczanie krawędzi progowaniem histerezy

Po poprzednim kroku na obrazie nadal znajdują się nieistotne krawędzie. W tym celu Canny wprowadził ideę progowania histerezy. Metoda ta wymaga 2 wartości progowych  $T_1, T_2$  takich, że  $T_1 < T_2$ . Jeżeli wartość gradientu w danym pikselu jest większa od  $T_2$ , to zaznaczono ten punkt jako krawędź. Jeśli tak się stało, to zaczęto proces śledzenia krawędzi — dla każdego sąsiada, którego wartość gradientu jest większa od  $T_1$  zaznaczono go jako krawędź. Jest ona wykonywana rekurencyjnie dla każdego zakwalifikowanego punktu.

Zamiast dokładnych wartości progowych można przekazać do funkcji 2 wartości —  $t_1, t_2$ , które są procentem liczby pikseli, które będą niedopuszczone jako krawędzie. Dla  $t_1 = 0.7, t_2 = 0.9$  dopuszczono tylko 10% pikseli jako te, które są większe od  $T_2$ . Podając  $t_1, t_2$  wyznaczono rozkład wartości gradientu w badanym obrazie, obliczono dystrybuantę  $F(x)$  i wybrano dla  $T_1$  ten argument, dla którego  $F(x) = t_1 * \text{liczbapikseli}$  i analogicznie dla  $T_2$ . W ten sposób wyznaczono progi do histerezy.

Operator Sobela [6] to metoda wyznaczania gradientu, a więc zarazem krawędzi zarówno w kierunku poziomym, jak i w pionowym. Dla każdego piksela przeprowadzono operację morfologiczną z następującymi maskami:

Maska rzędów			Maska kolumn		
-1	-2	-1	-1	0	1
0	0	0	-2	0	2
1	2	1	-1	0	1

### 3.3. OPRACOWANY ALGORYTM PÓŁAUTOMATYCZNY

Po wykonaniu tych operacji otrzymano wartości  $s_1$  i  $s_2$  odpowiednio dla maski rzędów i kolumn. Na podstawie tych danych otrzymano następujące informacje o gradiencie:

Wielkość gradientu    Kierunek krawędzi

$$\sqrt{s_1^2 + s_2^2} \qquad \tan^{-1} \left[ \frac{s_1}{s_2} \right]$$

Detektory krawędzi oparte na gradiencie, w tym operator Canny’ego są często używane. Ich główne zalety na podstawie [4] to:

- Dają dobre wyniki dla obrazów o dobrej jakości i bez szumów.
- Są wydajne - ich złożoność jest liniowa względem liczby przetwarzanych pikseli.
- Nie wymagają skomplikowanej sztucznej inteligencji do działania.

Zgodnie z [4] za ich główne wady można uznać:

- „Konieczność określenia rozmiaru maski i wartości progowej. Rozmiar maski znacząco wpływa na położenie miejsc, w których gradient przecina zera lub osiąga wartości maksymalne.
- Pomijanie narożników spowodowane faktem, że wartość 1D gradientu w narożnikach jest zazwyczaj mała.
- Operator pierwszej pochodnej wykrywa tylko schodkowe krawędzie.
- Duża wrażliwość na szum.”
- Na podstawie obserwacji działania algorytmu - rozmyte krawędzie często nie są wykrywane przez małe różnice w wartościach kolejnych sąsiadujących pikseli.

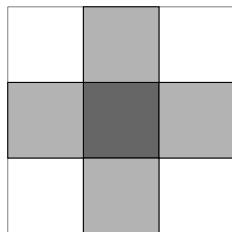
W ten sposób otrzymano macierz, gdzie każde pole w macierzy odpowiada pikselowi w wejściowej bitmapie — obrazie medycznym. Jeśli w komórce macierzy znajduje się 1, to w tym miejscu na bitmapie znajduje się krawędź, w przeciwnym przypadku 0. W ten sposób algorytm wykrył wszystkie znaczące krawędzie na bitmapie. Kolejnym krokiem przetwarzania było stworzenie grafu na podstawie wyżej wymienionej macierzy.

#### 3.3.2. Stworzenie grafu z bitmapy

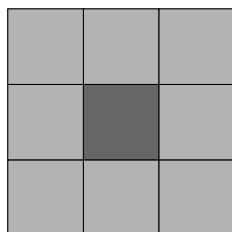
Na tym etapie algorytm potrzebuje następujących danych wejściowych:

- Macierz z wartościami logicznymi prawda/fałsz czy znajduje się w danym punkcie krawędź. Może to być także realizowane poprzez macierz wartości liczbowych.
- Punkty wybrane przez użytkownika aplikacji.

Przed rozpoczęciem działania algorytmu należy zapewnić łączność 4-krotną (ang. Pixel 4-connectivity) [7]. Jest ona zwana także sąsiedztwem von Neumanna. Przy łączności 4-krotnej sprawdza się tylko sąsiadów w poziomie lub pionie.



Dla łączności 8-krotnej sprawdza się wszystkich możliwych sąsiadów, także po przekątnej. Jest ona zwana także sąsiedztwem Moore'a lub otoczeniem Moore'a [8].



W przypadku zastosowania łączności 8-krotnej przy wyznaczaniu długości krawędzi musiano by zastosować metrykę Czebyszewa, która jest specjalnym przypadkiem odległości Minkowskiego. Jeśli zostanie łączność 4-krotna to długość krawędzi byłaby obliczana zgodnie z metryką miejską, zwaną też metryką Manhattan.

Metryka Manhattan w kontekście dalszego przetwarzania w celu wyszukiwania najkrótszych ścieżek w grafie jest bardziej adekwatna, ponieważ jest intuicyjna w wyznaczaniu odległości na obrazie płaskim w porównaniu do metryki Czebyszewa. W tym przypadku najlepsza byłaby tutaj metryka Euklidesa, lecz mamy do czynienia nie z kolejnymi punktami oddalonymi od siebie, a z sąsiadującymi pikselami. Ponadto w tym algorytmie istotne jest szybkie szacowanie odległości, czy też długości danej krawędzi.

Wykrywanie wierzchołków przy łączności 4-krotnej jest prostsze. Wystarczy zliczyć liczbę sąsiadów. Poniżej zakładamy, że piksel jest oznaczony jako krawędź w macierzy wejściowej. W zależności od liczby sąsiadów mamy następujące przypadki:

- 0 — wierzchołek izolowany,
- 1 — punkty końcowe (ang. endpixels),
- 2 — punkty łączące (ang. linkpixels), czyli fragmenty krawędzi,
- 3–4 — punkty węzłowe (ang. vertices), czyli punkty, od których odchodzą co najmniej 3 krawędzie.



### 3.3. OPRACOWANY ALGORYTM PÓŁAUTOMATYCZNY

W przypadku łączności 8-krotnej do detekcji wierzchołków należałoby stosować przekształcenia Hit-or-Miss z elementami strukturalnymi. Elementy strukturalne do wykrywania odpowiednich punktów są następujące:

- wierzchołek izolowany:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

- punkty końcowe (ang. endpixels):

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ z & z & z \end{bmatrix},$$

- punkty łączące (ang. linkpixels), czyli fragmenty krawędzi, posiadają dokładnie 2 sąsiadów, nie używamy przekształcenia Hit-or-Miss a tylko liczymy sąsiadów

- punkty węzłowe (ang. vertices), czyli punkty, od których odchodzą co najmniej 3 krawędzie:

$$\begin{bmatrix} z & 1 & z \\ z & 1 & z \\ z & z & 1 \end{bmatrix} \text{ lub } \begin{bmatrix} 1 & z & z \\ z & 1 & z \\ 1 & z & 1 \end{bmatrix},$$

Warto zauważyć, że te elementy strukturalne należy obracać o 90, 180 i 270 stopni. Za każdym razem trzeba wielokrotnie sprawdzać te same piksele. Ponadto należy sprawdzać 8, a nie 4 sąsiadów.

Kolejnym problemem jest fakt, że przy spójności 8-krotnej przekształcenie Hit-or-Miss może w najbliższym otoczeniu punktu krzyżowania się krawędzi oznaczyć kilka otaczających punktów, jako punkty węzłowe. Jest to złe rozwiązanie, ponieważ w ten sposób może nawet kilkukrotnie zwiększyć liczbę wierzchołków w grafie, co przełożyłoby się na niską wydajność algorytmu.

Ostatnim problemem z jakim należałoby się wiązać wybierając łączność 8-krotną jest fakt, że macierz wejściową dla tego etapu algorytmu należałoby poddać procesowi szkieletyzacji. Najlepiej byłoby w tym celu skorzystać z algorytmu KMM [9] lub K3M [10]. Te algorytmy musiałyby co najmniej raz przejrzeć całą macierz z wykrytymi krawędziami w optymistycznym przypadku.

Z wyżej wymienionych powodów zdecydowano się na łączność 4-krotną. Przygotowano i zaimplementowano algorytm tworzący graf z bitmapy, a jego pseudokod znajduje się poniżej.

- 1) MATRIX - macierz wejściowa z oznaczonymi krawędziami jako 1
- 2) foreach(punkt A taki, że MATRIX(A) == 1)
- 3) {

```

4)    if ( punkt A ma 1 sąsiada albo co najmniej 3 sąsiadów )
5)    {
6)        // (tzn. jest albo punktem końcowym albo węzłowym)
7)        wstaw punkt A do kolejki wierzchołków L_V
8)        while ( kolejka wierzchołków L_V niepusta )
9)        {
10)            weź wierzchołek V_1 z kolejki L_V
11)            usuń V_1 z kolejki L_V
12)            dodaj wierzchołek V_1 do grafu G
13)            foreach (punkt B sąsiadujący z V_1 )
14)            {
15)                if ( MATRIX(B) == 1 )
16)                {
17)                    stwórz nową krawędź E.
18)                    dodaj B do E
19)                    ustaw wierzchołek V_1 jako początek krawędzi E
20)                    dodaj krawędź E do kolejki przetwarzanych krawędzi L_E
21)                }
22)            while ( L_E niepusta)
23)            {
24)                weź krawędź E z L_E
25)                usuń E z kolejki
26)                stwórz kolejkę potencjalnych punktów krawędzi E, L_P
27)                weź punkt C z E
28)                usuń C z E
29)                dodaj punkt C do kolejki L_P
30)                while ( L_P niepusta )
31)                {
32)                    weź punkt D z L_P
33)                    usuń D z L_P
34)                    K = liczba sąsiadów D
35)                    if ( K == 0 lub K > 1 )
36)                    {
37)                        stwórz wierzchołek V_2, który znajduje się w D

```

### 3.3. OPRACOWANY ALGORYTM PÓŁAUTOMATYCZNY

```
38)                do listy krawędzi wierzchołka V_2 dodaj E
39)                do listy krawędzi wierzchołka V_1 dodaj E
40)                ustaw wierzchołek V_2 jako koniec krawędzi E
41)                dodaj krawędź E do grafu G
42)                dodaj wierzchołek V_2 do kolejki L_V
43)            }
44)        if ( K == 1 )
45)        {
46)            dodaj punkt D do E
47)            foreach ( punkt F sąsiadujący z D )
48)            {
49)                if ( MATRIX(F) == 1)
50)                {
51)                    dodaj F do L_P
52)                }
53)            }
54)        }
55)        MATRIX(D) = 1
56)    }
57) }
58) }
59) }
60) }
61) }
62) Zwróć graf G
```

Utworzony w ten sposób graf jest grafem nieskierowanym z wagami, gdzie wagi to liczba pikseli, czy też punktów należących do krawędzi. Graf ten może nie być spójny. Ponadto ten graf może nie zawierać wierzchołków, które pokrywają się z punktami wybranymi przez użytkownika.

Graf ten zazwyczaj jest rzadki, ponieważ liczba jego krawędzi jest rzędu liczby jego wierzchołków. Z uwagi na czasami bardzo dużą liczbę wierzchołków — nawet do kilkudziesięciu tysięcy — próba implementacji przy pomocy macierzy sąsiedztwa mogłaby spowodować zużycie całej możliwej pamięci operacyjnej. Dla 50 tysięcy wierzchołków program musiałby zadeklarować macierz sąsiedztwa zawierającą 2,5 miliarda komórek. Z tych powodów graf został zaimplementowany przy pomocy list sąsiedztwa.

W przyjętej implementacji każda krawędź zawiera dodatkowo informację o tym, jakie piksele należą do danej krawędzi w rzeczywistym obrazie.

### 3.3.3. Zapewnienie spójności grafu

W pierwszym kroku na tym etapie przetwarzania grafu są dodawane punkty wybrane przez użytkownika do grafu.

Graf, który uzyskano w poprzednim kroku może nie być spójny. Powoduje to fakt, że między punktami wybranymi przez użytkownika mogą nie istnieć ścieżki. W celu zapewnienia spójności grafu należy dodawać sztuczne krawędzie. Zostają one dodawane z większymi wagami, niż wynikałoby to w rzeczywistości z liczebności listy pikseli, które reprezentują, ponieważ ma to na celu używanie z większym priorytetem prawdziwych krawędzi, a nie sztucznych. W sytuacji gdy nie istnieje dobra ścieżka z prawdziwych krawędzi, to zostaną użyte krawędzie sztuczne. Duże wagi dodatkowo będą zmuszały algorytmy wyszukiwania najkrótszych ścieżek w grafie do minimalizowania długości takich fragmentów zawierających sztuczne krawędzie.

Do znajdowania spójnych składowych grafu najczęściej używa się albo algorytmu przeszukiwania grafu w głąb (ang. Depth-first search, w skrócie DFS) lub przeszukiwania grafu wszerz (ang. Breadth-first search, w skrócie BFS) [11]. Użyto algorytmu przeszukiwania wszerz, opierając się na przykładowej implementacji w materiałach [14]. Wykorzystano `Queue<T>`, a zatem kolejkę, więc jest to przeszukiwanie wszerz. Złożoność obliczeniowa tego algorytmu to  $O(|E|)$ .

Po wyznaczeniu spójnych składowych grafu dla każdej pary składowych найдowano taką parę wierzchołków, że odległość między nimi jest minimalna. Jeśli ta odległość była mniejsza niż maksymalna odległość między dwoma dowolnymi punktami zaznaczonymi przez użytkownika, to była dodawana sztuczna krawędź o wadze 2,5 razy większej niż odległość wynikająca z metryki Manhattan pomiędzy tymi dwoma wierzchołkami.

Przykład zasady działania tej operacji przedstawia poniższy pseudokod:

```

1) foreach (Spójna składowa grafu s1)
2) {
3)     foreach (Spójna składowa grafu s2, różna od s1 i nie przetwarzana
        wcześniej jako s1)
4)     {
5)         Wybierz wierzchołek v1 z s1 i v2 z s2 takie, że odległość pomiędzy
            nimi jest najmniejsza ze wszystkich takich par
6)         Dodaj sztuczną krawędź pomiędzy v1 i v2
7)     }
```

8) }

W ten sposób osiągnięto spójny graf, który zawiera punkty dodane przez użytkownika.

### 3.3.4. Wyszukanie najkrótszych ścieżek w grafie

Na tym etapie została zapewniona spójność grafu. Z całą pewnością istnieje ścieżka pomiędzy punktami wybranymi przez użytkownika, te punkty są osiągalne. Pozostało wybrać najkrótszą ścieżkę łączącą kolejne te punkty. Założono, że jest dana lista punktów wybranych przez użytkownika i zawiera ona kolejne punkty, tzn. pierwszy należy połączyć z drugim, drugi z trzecim, ..., ostatni z pierwszym.

Warto zaznaczyć, że wyszukiwano ścieżki o minimalnej wadze, czy też o minimalnym koszcie. Z uwagi na fakt, że wagi w grafie są ściśle związane z odległościami to używane jest sformułowanie szukania najkrótszych ścieżek. Wagi w tym grafie są nieujemne. Może istnieć w tym grafie kilka ścieżek z jednego punktu do drugiego o tym samym koszcie, więc algorytm kończy swoje działanie na tym etapie, gdy znajdzie jedną z nich. W tym grafie nie występują krawędzie wielokrotne. Ten graf nie zawiera cykli własnych.

Do wyszukiwania najkrótszych ścieżek w grafie po zgłębieniu literatury [11] i [14] były rozważane do użycia 2 algorytmy. Był to algorytm Dijkstry i A\*. Do algorytmu A\* była rozważana heurystyka w postaci odległości miejskiej, Manhattan z wierzchołka  $v$  do celu  $t$ , ozn.  $h(v)$ .

Zgodnie z [14] „Funkcja  $h$  musi spełniać następujące warunki:

- musi być oszacowaniem dolnym, czyli dla każdego wierzchołka  $v$   $h(v) \leq$  odległość  $v$  od celu  $t$ ,
- musi być monotoniczna, czyli dla dowolnej krawędzi  $\langle u, v \rangle$   $h(u) \leq \text{waga} \langle u, v \rangle + h(v)$ .”

Heurystyka w postaci metryki Manhattan spełnia te wymagania. Z tego powodu może zostać wykorzystany algorytm A\*. Porównując złożoności algorytmu A\* i Dijkstry w [14] możemy zauważyć, że algorytm A\* w pesymistycznym przypadku ma taką samą złożoność jak algorytm Dijkstry, czyli  $O(|E| * \log(|V|))$  z kolejką priorytetową dla grafów rzadkich, a  $O(|V|^2)$  nie wykorzystując kolejki priorytetowej dla grafów gęstych. W praktyce dla grafów rzadkich nie ma potrzeby rozważania znacznej części wierzchołków, co poprawia złożoność średnią. Wynosi ona wtedy  $O(|E|)$ .

Do implementacji wyszukiwania najkrótszych ścieżek w grafie na podstawie wyżej wymienionych wniosków został wykorzystany algorytm A\* z heurystyką w postaci metryki Manhattan. Przykładowy pseudokod tego algorytmu można znaleźć w [14] i jest on następujący:

```

1) CLOSE = 0 // zbiór (z szybkim sprawdzeniem przynależności)
2) OPEN = {s} // kolejka priorytetowa
3) // priorytety - sumy odległości wierzchołków od źródła
4) // i oszacowań odległości tych wierzchołków od celu
5) odległość[s] = 0
6) while ( OPEN niepusty )
7) {
8)     u = wierzchołek należący do OPEN taki, że
9)         odległość[u] + oszacowanie[u,t] <=
10)        odległość[w] + oszacowanie[w,t]
11)        dla wszystkich w należących do OPEN
12)    usuń u z OPEN
13)    wstaw u do CLOSE
14)    if ( u == t ) break
15)    foreach ( wierzchołek w sąsiadujący z u taki, że w należy do CLOSE )
16)    {
17)        if ( w należy do OPEN )
18)        {
19)            odległość[w] = nieskończoność
20)            wstaw w do OPEN
21)        }
22)        if ( odległość[w] > odległość[u] + waga<u,w> )
23)        {
24)            odległość[w] = odległość[u] + waga<u,w>
25)            aktualizacja priorytetu wierzchołka w (w OPEN)
26)            poprzedni[w] = u
27)        }
28)    }
29) }

```

Aby maksymalnie dobrze wykorzystać niską złożoność obliczeniową algorytmu A\* należało wykorzystać dobre struktury danych do tego algorytmu. Zbiór CLOSE wymagał szybkiego sprawdzania przynależności. Po przeanalizowaniu rekomendowanych struktur danych [15] został użyty `HashSet<T>`.

Zbiór Open oferował zastosowanie znacznie bardziej finezyjnych struktur danych. Wymagane

było od niego, aby był kolejką priorytetową, czyli aby był sortowany po priorytetach będącymi sumą odległości wierzchołków od źródła i oszacowań odległości tych wierzchołków od celu. Warto zauważyć, że oprócz sortowania często są wstawiane do niego wierzchołki, pobierane, usuwane i modyfikowane wartości priorytetu. Bardzo często jest sprawdzana przynależność i jest wyszukiwanie według klucza.

W [12] na stronie 317 znajduje się tabela 7.1. Przedstawia ona porównanie czasów różnych operacji dla różnych klas słownikowych zaimplementowanych w technologii .NET. Najlepszą strukturą danych byłoby `SortedDictionary<K,V>`, która jest implementowana przez drzewo czerwono-czarne gdyby nie fakt, że jest sortowane po kluczu, a nie tak jak jest potrzebne w tym przypadku sortowanie po wartości. Poszukiwano zatem struktury danych, która sortuje po wartościach i ma łatwy dostęp przez klucz.

W naszym rozwiązaniu została zaimplementowana struktura danych opierająca się na dwóch podstrukturach - zaimplementowanej kolejki priorytetowej poprzez listę, oraz słownik `Dictionary<K,V>` z technologii .NET, który opiera się o Tablicę skrótów. Zaimplementowana lista miała następującą strukturę:

```
public class MySortedListElement
{
    public Vertex Key;
    public double Value;
    public MySortedListElement next;
    public MySortedListElement previous;
}
```

Struktura ta miała zaimplementowane sortowanie przez wstawianie, zatem modyfikacja tylko jednego elementu wymagała w pesymistycznym przypadku tylko  $O(n)$  porównań. Wstawianie nowego elementu ma złożoność pesymistyczną  $O(n)$ , usuwanie  $O(1)$ . Zbadanie przynależności po kluczu to  $O(n)$  i nie zaleca się tego robić.

Niezależnie od tej struktury jest przechowywana druga struktura danych, `Dictionary<K,V>`, gdzie kluczami są wierzchołki, a wartości to referencje na elementy tej listy. Zmienne odpowiedzialne za przechowanie kolejki priorytetowej OPEN zostały zadeklarowane w sposób następujący:

```
Dictionary<Vertex, MySortedListElement> OpenDictionary =
    new Dictionary<Vertex, MySortedListElement>();
MySortedList OpenList = new MySortedList();
```

W ten sposób w połączeniu tych dwóch podstruktur otrzymujemy strukturę danych o następujących własnościach:

- dodawanie w czasie  $O(n)$ ,
- wyszukiwanie po kluczu w czasie  $O(1)$ ,
- wybierz najmniejszy element według wartości w czasie  $O(1)$ ,
- usuwanie w czasie  $O(1)$
- modyfikowanie jednego elementu w czasie  $O(n)$ .

Algorytm A\* jest uruchamiany dla każdego punktu wybranego przez użytkownika. Wyszukuje on najkrótszą ścieżkę do kolejnego punktu wybranego przez użytkownika. Po wszystkich obliczeniach obrys składa się z poszczególnych ścieżek. Są one konsolidowane i zwracane jako lista pikseli na podstawie danych z krawędzi o listach pikseli, z których jest zbudowana krawędź. W ten sposób jest wykrywany obrys pomiędzy punktami zaznaczonymi przez użytkownika.

W sytuacji, gdy pomiędzy różnymi wierzchołkami nie istnieją prawdziwe krawędzie, albo istnieją tylko w wybranym fragmencie ścieżki łączącej te dwa wierzchołki, to algorytm w miarę możliwości będzie starał się używać prawdziwych krawędzi, dzięki odpowiedniej wadze krawędzi sztucznych. Dzięki sztucznym krawędziom na pewno istnieje droga między kolejnymi punktami, zatem algorytm z całą pewnością zwróci poprawny wynik. Co najwyżej będzie to wielokąt z punktami wybranymi przez użytkownika.

Algorytm dla sztucznych krawędzi generuje listę pikseli algorytmem Bresenhama [16]. Jest to algorytm, który dodaje w najbardziej optymalny sposób krawędzie. W celu zapewnienia łączności 4-krotnej jest przeprowadzana operacja morfologiczna z wykrywaniem 2 pikseli po przekątnej z maską 4-pikselową. Gdy zostaną takie piksele wykryte, to na jednym z rogów jest dodawany piksel w celu zapewnienia łączności 4-krotnej.

### 3.3.5. Optymalizacja

Większość plików medycznych ma rozdzielczości rzędu kilkaset na kilkaset pikseli, a więc na całym obrazie znajduje się kilkaset tysięcy pikseli. Zdarzają się też pliki w znacznie większej rozdzielczości, a takimi jest między innymi mammografia i zdjęcie rentgenowskie. Na tych obrazach znajduje się do kilkudziesięciu milionów pikseli. W celu sprawnego przetwarzania tych informacji potrzebne były optymalizacje do algorytmu. Zostały dodane następujące usprawnienia:

#### 1. Zmniejszenie rozmiaru obrazu roboczego

W pierwszym kroku zmniejszono rozmiar obrazu roboczego. Wyznaczono najmniejszy prostokąt, w którym mieszczą się wszystkie punkty zaznaczone przez użytkownika. Powiększono ten prostokąt o marginesy w taki sposób, by wyjściowy prostokąt nadal był w środku,



### 3.4. MODUŁ OBLICZEŃ STATYSTYK

a pole powiększonego było 4-krotnie większe. Innymi słowami — dodano marginesy po 50% szerokości / wysokości do każdego wymiaru. Jeśli rozmiar powiększonego prostokąta jest większe niż obrazu, to nie jest zmieniany obraz roboczy. Optymalizując w ten sposób w sytuacji, gdy użytkownik stworzył mały obrys, to rozmiar przetwarzanego obrazu od momentu operatora Sobela jest już najczęściej kilkukrotnie mniejszy, czasami nawet kilkadziesiąt razy. W sytuacji gdy obrys zajmuje prawie cały obraz medyczny, to nadal musi być przetwarzany cały obraz.

#### 2. Sięganie do pamięci zamiast to bitmapy

W celu przyspieszenia działania operatora Sobela i liczenia statystyk zamiast sięgać do bitmapy metodą `.GetPixel()` na platformie .NET bitmapa została zapisana do tablicy bajtów i był z niej bezpośredni dostęp. Została wykorzystana do tego funkcja `.LockBits()`. Zysk na tej operacji był kilkudziesięciokrotny. Na późniejszych etapach algorytmu pracowano na macierzy, która nie była już bitmapą, więc operacje te wykonywały się znacznie szybciej.

#### 3. Dzielenie zbyt długich krawędzi

W sytuacji, gdy wczytane krawędzie do grafu są bardzo długie, to może zdarzyć się taka sytuacja, że punkt zaznaczony przez użytkownika znajduje się bardzo blisko krawędzi, ale daleko od punktu początkowego lub końcowego krawędzi. W tym celu jest wprowadzony mechanizm dzielenia zbyt długich krawędzi na kilka krótszych.

#### 4. Problem z ilością punktów

Dla każdego punktu jest uruchamiany algorytm  $A^*$ , zatem w sposób liniowy od ilości punktów zależy liczba wywołań algorytmu  $A^*$ . Nie można wyznaczyć, czy w złożoności średniej całego algorytmu algorytm jest zależny liniowo od ilości punktów, czy w sposób logarytmiczny, czy też w sposób stały.

### 3.4. Moduł obliczeń statystyk

Nasz system zapewnia liczenie statystyk dotyczących obrysu. Statystyki są identyczne zarówno dla obrysów manualnych i półautomatycznych. Następujące statystyki są prezentowane użytkownikowi:

- Histogram,
  - Wartość maksymalna,
  - Wartość minimalna,

- Wartość średnia,
- Środek ciężkości,
- Długość obrysu w  $mm$ ,
- Długość obrysu w pikselach,
- Pole powierzchni w  $mm^2$ ,
- Liczba pikseli wewnątrz obrysu.

Do obliczenia histogramu jest wymagany punkt wewnętrzny obrysu. Środek ciężkości obrysu nie zawsze musi znajdować się wewnątrz obrysu. W przypadku zastosowania algorytmu scan-linii dla obrysu manualnego nie jest dana informacja o logicznych wartościach gdzie jest krawędź, więc nie można wykryć np. krawędzi poziomych. Algorytm scan-linii lepiej nadaje się w przypadku gdy wypełniamy wielokąt, a nie obrys.

Z tego powodu wybrano algorytm Flood Fill, czyli rozlewania się rekurencyjnego zliczonych pikseli od punktu wewnętrznego. Właśnie z tego powodu od użytkownika jest wymagane podanie dodatkowej informacji, jaką jest punkt wewnętrzny obrysu. Założono, że użytkownik zaznaczy go poprawnie. W sytuacji, gdy zostanie podany błędny punkt to ten algorytm policzy to co znajduje się na zewnątrz obrysu, a nie to co jest wewnątrz.

## 4. Przeprowadzone eksperymenty

### 4.1. Zbiór testowy

### 4.2. Wydajność algorytmu półautomatycznego

### 4.3. Analiza wyników i wnioski

## 5. Podsumowanie

### 5.1. Napotkane problemy i ograniczenia

### 5.2. Możliwości dalszego rozwoju

## Bibliografia

- [1] Nowacki R., Plechawska-Wójcik M.: Analiza porównawcza narzędzi do budowania aplikacji Single Page Application — AngularJS, ReactJS, Ember.js, *Journal of Computer Sciences Institute 2, Politechnika Lubelska, Instytut Informatyki*, Lublin, 2016, 98–103
- [2] Microsoft Corporation: Dokumentacja platformy .NET. Oficjalna strona: <https://docs.microsoft.com/pl-pl/dotnet/> [Dostęp 22 stycznia 2019]
- [3] Microsoft Corporation: Informacje o platformie .NET Core. <https://docs.microsoft.com/pl-pl/dotnet/core/about> [Dostęp 22 stycznia 2019]
- [4] Kronis K., Uhanova M.: Performance Comparision of Java EE and ASP.NET Core Technologies for Web API Developmnet. *Applied Computer Systems 23, Riga Technical University*, Ryga, 2018, 37–44
- [4] Cytowski J., Gielecki J., Gola A.: Cyfrowe przetwarzanie obrazów medycznych: Algorytmy. Technologie. Zastosowania. *Akademicka Oficyna Wydawnicza EXIT*, Warszawa, 2008, 88–94.
- [5] Canny J. F.: Finding Edges and Lines in Images. *Technical report no. 720, Massachusetts Institute of Technology (MIT)*, Cambridge, Massachusetts, USA, 1983.
- [6] Sobel I., Feldman G.: An 3x3 Isotropic Image Gradient Operator for Image Processing. *Presentation at Stanford Aartificial Intelligence Project (SAIL) in 1968*, 2014
- [7] Rosenfeld A., Kak A. C.: Digital Picture Processing, *Academic Press, Inc.*, Nowy Jork, 1982
- [8] Weisstein, Eric W.: Moore Neighborhood. *From MathWorld—A Wolfram Web Resource*. <http://mathworld.wolfram.com/MooreNeighborhood.html> [Dostęp 22 stycznia 2019]
- [9] Saeed K., Rybnik M., Tabędzki M., Adamski M.: Algorytm do Ścieniania Obrazów: Implementacja i Zastosowania *Zeszyty Naukowe Politechniki Białostockiej 2002 Informatyka - Zeszyt 1*, Białystok, 2002

- [10] Saeed K., Tabędzki M., Rybnik M., Adamski M.: K3M: A Universal Algorithm for Image Skeletonization and a Review of Thinning Techniques *International Journal of Applied Mathematics and Computer Science*, 2010, 20(2) Białystok, 2010, 317–335
- [11] Sedgewick R., Wayne K.: Algorytmy Wydanie IV, *Helion*, Gliwice, 2012, 526–706
- [12] Albahari J., Albahari B.: C 6.0 w pigułce, *Helion, O'Reilly Media, Inc.*, Gliwice, 2016, page–page
- [13] Hart P. E., Nilsson N. J., Raphael B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths *IEEE Transactions on Systems Science and Cybernetics* 4(2), 1968, 100–107
- [14] Bródka J.: Wykłady z przedmiotu Algorytmy i Struktury Danych 2 *Politechnika Warszawska, Wydział Matematyki i Nauk Informacyjnych* Materiały dostępne na stronie: <http://mini.pw.edu.pl/brodka/ASD2.html> [Dostęp 22 stycznia 2019]
- [15] Microsoft Corporation: Dokumentacja rekomendowanych struktur danych platformy .NET. Oficjalna strona: <https://docs.microsoft.com/pl-pl/dotnet/standard/collections/> [Dostęp 22 stycznia 2019]
- [16] Bresenham J. E.: Algorithm for computer control of a digital plotter. *ICM System Journal* 4(1) 1965

## Instrukcja instalacji

## Instrukcja użytkowania



## Wykaz symboli i skrótów

nzw.    nadzwyczajny

\*       operator gwiazdka

~       tylda

Jak nie występują, usunąć.

## Spis zawartości załączonej płyty CD

## Spis załączników

1. Załącznik 1
2. Załącznik 2
3. Jak nie występują, usunąć rozdział.